Continuous Assessment Project

The task is to pick a dataset, create a deep learning generation model, evaluate the model and present a report.

1. Pick a corpus

It is up to the team to decide which dataset to train the model. If you are clueless you can take a look at:

- Wikipedia corpus:
  - https://www.kdnuggets.com/2017/11/building-wikipedia-text-corpus-nlp.html
- Trump/Hilary corpus:
  - https://www.kaggle.com/benhamner/clinton-trump-tweets
  - https://www.kaggle.com/kingburrito666/better-donald-trump-tweets
- Quotes
  - https://github.com/alvations/Quotables

For tokenizers, use simple ones or go down to the character level. If you want to challenge yourself, you can try the fancy subwords models.

Hint:

- Generally, look for one that requires little cleaning =)
- If you have an idea what kind of corpus you like but dont know where to find, come talk to me during break or after class

2. Pick a model

It's up to the team to decide which model to implement for generation, go as fancy as you want. The quality of the model won't be an assessment as much as the sound-ness of how you implemented the model.

I strongly suggest Feed-Forward Language Model or RNN Language Model, but don't stop there if you really want to challenge yourself and write a Transformer model or create a never seen before model for the task.

You can also use external libraries to build a model but you have to add the instructions on to run the generation if you're not doing a simple model that I can run on a jupyter notebook.

References / Hint:

- Character RNN Generation (PyTorch Docs) https://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html
- Character RNN Generation (NLP for PyTorch) https://github.com/joosthub/PyTorchNLPBook/tree/master/chapters/chapter_7/7_3_surname_generation
- Spro's Char RNN https://github.com/spro/practical-pytorch/tree/master/char-rnn-generation

- Alvations' Word RNN
  http://bit.ly/ANLP-Session5-WordRNN

3. Evaluate the model

There's might be a couple of ways to do the evaluation. But this might be the simplest

> **Turing Test:** E.g. you can also sample 25 generated sentences and 25 sentences from the test set, then assign a score humanly the sentences from 1-10 to see whether they are human. See which one wins.

If you're lazy to do some annotations for turing test.

> **Perplexity:** See Lecture 5 slides on how to compute perplexity, compute the perplexity on all data splits (training, validation and test set)

I would consider this hack okay (for CA):

> Calculating perplexity is 2^entropy, usually training on cross-entropy should report the loss. Sometimes you forward propagate the final model through each data split and then run the criterion for predictions against the actual data on all batches and average the losses across batches.

4. Write it all up

Summarize your project and describe the results in a paper, try your best to use one of these template from

- ACL LaTeX files
- ACL Microsoft Word files
- ACL Overleaf
- NIPS LaTeX
- NIPS Overleaf

If you have issues using these templates, talk to me and we'll figure out something else.

There's no need for much scientific references but use appropriately.

Deliverables

1. **Code to train the model** (good to have some comments or markdown descriptions if you're using a jupyter notebook)
    - A jupyter notebook to train the model and **show how the mode save** would be sufficient
    - If you have .py files or used a library to accomplish your project give a reference and tell me **how your run the code to create and save your model**

2. **Code to run the generation**
    - A jupyter notebook with some generated examples (you can use the same as the training one, use the Markdown descriptions to clearly **say this is where the model can be loaded and generation function**)
    - If you have .py files or used a library to accomplish your project, **add a documentation as to how to load you model and generate a couple of sentences**

3. **A trained model** (select only one best from all the models you've trained)

4. **The report**
    - Write an introduction to define the generation task. Define what is the task of language modelling and text generation (Hint: Wikipedia, or papers links in the slides)
    - Describe the model you use for the task and (i) how is the model evaluated, (ii) which model is chosen as the best model and why. Show some scores, training loss, perplexity or turing test results.
    - Present some example generations.
    - Conclude with what you've found to be interesting.

Frequently Asked Questions

Can I just copy and paste code?

Yes, we're testing you on what you've learnt here, just make sure what you copy and paste runs, saves a model and meet the objectives of the project. No deduction of points will be given if the model is simple, as long as it's trains and produces a model that can generate, it's okay.

Do I have to write from scratch? Can I use other tools to do the generation?

No, you don't have to write from scratch but you should have learnt all the tools you needed to do so and it might be easier than using other people's tool.

Yes, you can use other libraries/tools to build the model if so you would have to describe the model in greater details esp. if the model is not simply one layer of RNN or FFN.

If you have chosen to use a library, these are the assessment criteria for the code section:

- Which library did you use? Why choose this library?
- Have you documented how to run the code to produce the model you've submitted?
- Which model/architecture did you choose from the library? Choice of layers and cost function for the architecture you chose
- Describe how was the training done by the library.
- How are the hyperparameters tuned?
- Are the "Nuts and Bolts" topics applied in your model?

My model don't train well enough and the results looks jibberish. Is it okay?

Yes, it is okay if your final submitted model produces jibberish but you still have to report what hyperparameters you've tried and whenever possible why. The results of the output matters little more than how you reach there.